余論理式，再帰型，循環的言語

久木田水生

minao.kukita@gmail.com

科学哲学コロキアム
2011 年 7 月 30 日

背景:

- Greg Restall の**余論理式**.
- 論理と計算の**対応とギャップ**

目的:

- Restall の余論理式のアイディアを紹介し，それが直観的に何であるか，どのような点で有用であるかを考える.
- 余論理式のアイディアを改良する.
- 余論理式に対してどのような証明論が適切であるかを考える.
- 余論理式と自己言及的言明の関係について論じる.

概要

1. 余論理式
2. 論理と計算の関係
3. 再帰型としての余論理式
4. 証明体系
5. 循環的言語としての余論理式

1 余論理式

2 論理と計算の関係

3 再帰型としての余論理式

4 証明体系

5 循環的言語としての余論理式

ストリーム

余論理式は計算機科学における**ストリーム**の概念の応用である.
ストリームに関しては次の三つの観点で捉えることが出来る.

- 集合論的（静的）
- 計算的（動的）
- 圏論的（両方）

ストリーム：集合論的観点から

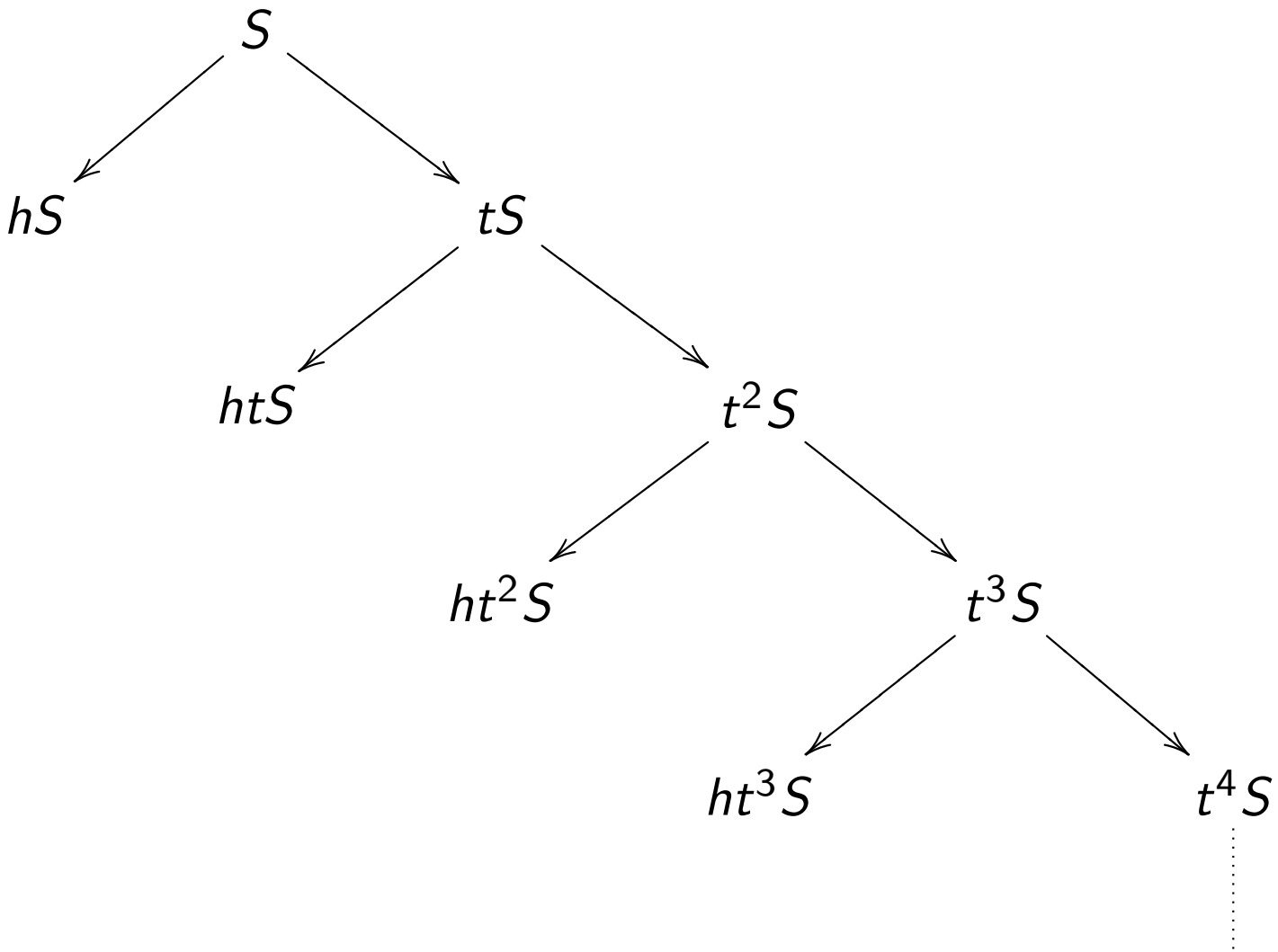
Definition

集合 A が与えられたとき，$Stream(A) = A^{\omega}$ $(= \prod_{n \in \omega} A = (\omega \to A))$.

$Stream(A)$ は次の等式を満たすことに注意せよ.

$$Stream(A) \simeq A \times Stream(A)$$

従って任意の $S \in Stream(A)$ は $a \in A$ と $S' \in A$ のペア (a, S') として表すことが出来る．このとき a を S の**ヘッド**，S' を S の**テール**と呼び，それぞれ $\mathbf{head}(S)$，$\mathbf{tail}(S)$ によって表す.



$h =$ **head** $\quad t =$ **tail**

ストリームの余帰納的定義

$Stream(A)$ は次のようにも定義される.

1. $S \in Stream(A)$ ならばある $a \in A$ と $S' \in Stream(A)$ に対して $s = (a, S')$.
2. 任意の集合 X と $x \in X$ に対して，ある $a \in A$ と $x' \in X$ が存在して $x = (a, x')$ が成り立つならば $X \subseteq Stream(A)$.

このような定義を**余帰納的定義**と呼ぶ.

参考：帰納的定義

データ型 A 上のリストの集合 $List(A)$ の帰納的定義と比較しよう．
$List(A)$ は次のように定義される．

1. $() \in List(A)$.
2. $a \in A$, $L \in List(A)$ ならば $(a, L) \in List(A)$.
3. 任意の集合 X に対して，$() \in X$ でありかつすべての $a \in A$ とすべての $x \in X$ に対して $(a, x) \in X$ が成り立つならば $List(A) \subseteq X$.

ストリームの定義

A 上のストリームを定義するにはヘッドとテールを指定すれば良い.
例えば

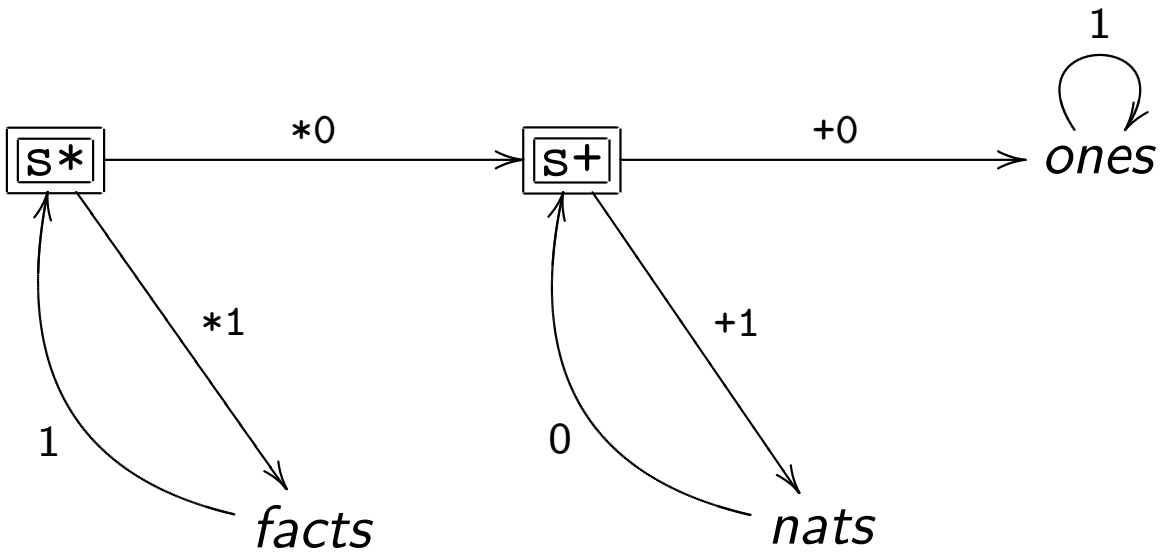
$$\mathtt{ones} = (1 : \mathtt{ones})$$

によって $(1, 1, 1, \dots)$ というストリームを定義できる.

例

名前	定義	出力
`ones`	`(1:ones)`	$1, 1, 1, 1, 1, 1, \ldots$
`nats`	`(0:(s+ ones nats))`	$0, 1, 2, 3, 4, 5, \ldots$
`facts`	`(1:(s* (`**`tail`**` nats) facts))`	$1, 1, 2, 6, 24, 120, \ldots$
`Fibs`	`(1:1:(s+ Fibs (`**`tail`**` Fibs)))`	$1, 1, 2, 3, 5, 8, \ldots$

ただし `s+`と `s*`はそれぞれ，二つのストリームの対応する構成要素同士を加えるあるいは掛けた値のストリームを出力する演算.


1
S*
*0
S+
+0
ones
*1
+1
1
0
facts
nats

ストリーム：計算的観点から

Definition

データ型 A 上のストリームとは，マシン $\mathcal{M} = (Q, q_0, \delta)$ である．ただしここで：

- Q はマシンの内部状態の集合，
- $q_0 \in Q$ は初期状態，
- $\delta : Q \to A \times Q$ は状態 q から原子データ a と状態 q' のペアへの関数．

$\delta(q) = \langle a, q' \rangle$ が意味するのは，状態 q にある $\mathcal{M}$ は a を出力して，次の状態 q' に移行するということである．このとき a と q' をそれぞれ q の **head**，**tail** と呼ぶ．特に q_0 の head と tail を，$\mathcal{M}$ の head と tail と呼ぶ．

注意

- A 上の任意のストリーム $\mathcal{M} = (Q, q_0, \delta)$ と $q \in Q$ に対して，$\mathcal{M}' = (Q, q, \delta)$ もまたストリームである．
- 逆に A 上の任意のストリーム $\mathcal{M} = (Q, q_0, \delta)$ と $a \in A$ に対して，$\mathcal{M}' = (Q \cup p, p, \delta_{p \mapsto (a, q_0)})$ もまた A 上のストリームである．ただしここで $p \notin Q$，$\delta_{p \mapsto (a, q_0)}$ は

$$\delta_{p \mapsto (a, q_0)}(q) = \begin{cases} (a, q_0) & q = p \text{ のとき} \\ \delta(q) & \text{それ以外のとき} \end{cases}$$

  によって定義される関数．
- このような $\mathcal{M}'$ を $(a, \mathcal{M})$ と表すことにしよう．
- 従って私たちは任意のストリーム $\mathcal{M} = (Q, q_0, \delta)$ を $(a : \mathcal{M}')$ として表すことが出来る．ただしここで $\delta(q_0) = (a, q_1)$，$\mathcal{M} = (Q, q_1, \delta)$ とする．

例

データ型として自然数 N を考える.

$$\begin{aligned} Q &= \{q\} \\ q_0 &= q \\ \delta(q) &= (1, q) \end{aligned}$$

によって定義される $\mathcal{M} = (Q, q_0, \delta)$ は $1,\ 1,\ 1,\ 1,\ 1, \ldots$ という出力を出し続けるマシンである.

ストリーム：圏論的観点から

Definition

$Stream(A)$ 終 F_A 余代数，すなわち圏 F_A-**CoAlg** における終対象である．ただし $F_A : \mathbf{Set} \to \mathbf{Set}$ は

$$F_A X = A \times X, \quad F_A f = \langle 1_A, f \rangle$$

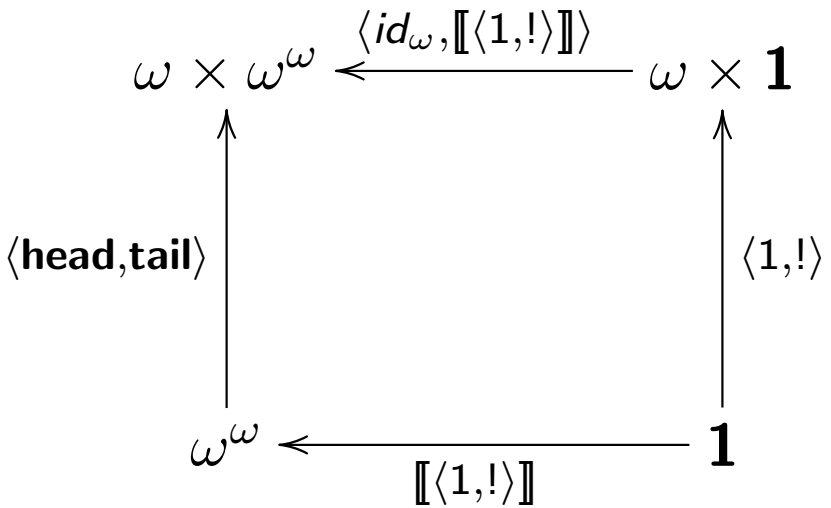
によって定義される関手であり，F_A-**CoAlg** は次の対象と射からなる圏：

- 対象：**Set** における関数 $f : X \to F_A X$.
- 射：$\mathbf{dom}(f)$ から $\mathbf{dom}(g)$ への関数 α で，$g \circ \alpha = F_A \alpha \circ f$ を満たすものを f から g への射とする．

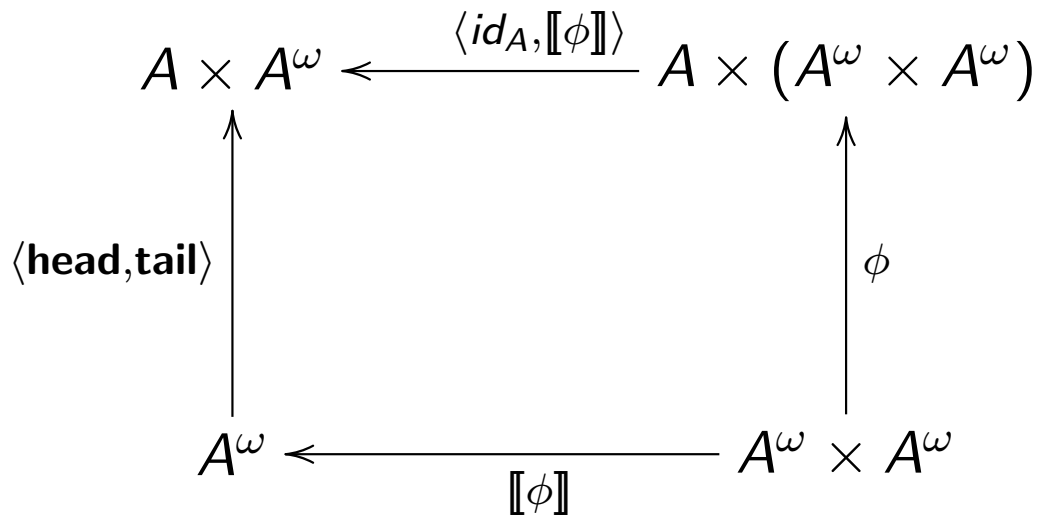
具体的に言えば $Stream(A) = \langle \mathbf{head}, \mathbf{tail} \rangle : A^\omega \to A \times A^\omega$.

$Stream(A)$ の要素や演算の多くはこの終対象への媒介射として定義できる．たとえば

$$\texttt{ones}\colon Stream(\omega)$$



$$\texttt{interleave}\colon \mathit{Stream}(A) \times \mathit{Stream}(A) \to \mathit{Stream}(A)$$



ただし $\phi = \langle \mathbf{head} \circ \mathbf{fst}, \langle \mathbf{snd}, \mathbf{tail} \circ \mathbf{fst} \rangle \rangle$.

余論理式（cf. Restall, forthcoming）

n 項結合子の集合 Σ_n（$n \geq 0$）が与えられているとしよう（$n = 0$ の場合は原子文の集合）．$\Sigma = \coprod_{n \geq 0} \Sigma_n$ とする．

Definition

Σ 上の余論理式の集合 T_Σ は以下の条件を満たす集合として定義される．

1. $\gamma \in T_\Sigma$ ならば，ある $\sigma \in \Sigma_n$ と $\gamma_1, \gamma_2, \ldots, \gamma_n \in T_\Sigma$ に対して $\gamma = \langle \sigma : \langle \gamma_1, \gamma_2, \ldots, \gamma_n \rangle \rangle$．
2. 任意の集合 X と $x \in X$ に対して，ある $\sigma \in \Sigma_n$ と $x_1, x_2, \ldots, x_n \in X$ が存在して $x = \langle \sigma : \langle x_1, x_2, \ldots, x_n \rangle \rangle$ が成り立つならば $X \subseteq T_\Sigma$．

$\gamma = \langle \sigma : \langle \gamma_1, \gamma_2, \ldots, \gamma_n \rangle \rangle$ であるとき，σ を γ の**ヘッド**，$\gamma_k (0 \leq k \leq n)$ を γ の**テール**と呼び，それぞれ $\mathbf{head}(\gamma), \mathbf{tail}_k(\gamma)$ によって表す．

余論理式

- 直観的には余論理式は，各ノードが $\mathscr{S}$ の要素でラベル付けされた無限木である．ただし各ノードが持つ子ノードの個数はそのノードがラベルとして持つ結合子のアリティに等しい．
- Σ にはアリティ 0 の結合子（つまり原子式）も含まれるため，余論理式は有限木である場合もある．このとき余論理式は通常の論理式である．

余論理式：別な定義

Definition

余論理式はマシン $\mathcal{M} = (Q, q_0, \delta)$ である．ただし

- Q は状態の集合，
- q_0 は初期状態
- $\delta : Q \to \Sigma \times \coprod_{n \geq 0} Q^n$ は状態 q から列 $(\sigma, p_1, p_2, \ldots, p_n)$ への関数．ただしここで $\sigma \in \Sigma_n$，$p_k \in Q$

$\delta(q_0) = (\sigma, p_1, p_2, \ldots, p_n)$ であるとき，σ を $\mathcal{M}$ の**ヘッド**，$p_k (1 \leq k \leq n)$ を**テール**と呼び，それぞれ $\mathbf{head}(\mathcal{M})$, $\mathbf{tail}_k(\mathcal{M})$ によって表す．

$\mathcal{M}$ が**有限**であると言われるのは Q が有限の場合である．

余論理式：別の定義

Definition

余論理式は F_Σ-**CoAlg** における終対象である．ただし $F_\Sigma : \mathbf{Set} \to \mathbf{Set}$ は次のように定義される関手：

$$F_\Sigma X = \coprod_{n \geq 0} \Sigma \times X^n \qquad F_\Sigma f = \coprod_{n \geq 0} \langle id_\Sigma, f^n \rangle$$

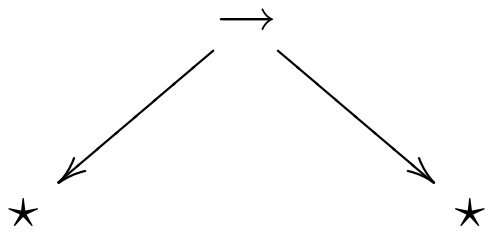
具体的にははこれはは $\coprod_{n\geq 0} \langle \mathbf{head}, \mathbf{tail}_1, \ldots, \mathbf{tail}_n \rangle : \coprod_{n \geq 0} (T_\Sigma \to \Sigma \times T_\Sigma^n)$. ただし T_Σ は上記の無限木の集合.

例

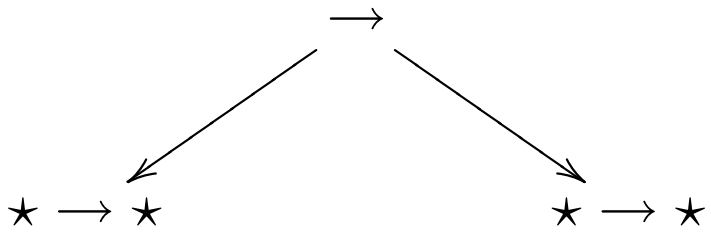
$\to$ と $\wedge$ は 2 項結合子とする．このとき以下は余論理式である．

- $Q = \{\star\},\ q_0 = \star,\ \delta(\star) = \langle \to, \star, \star \rangle$.
- $Q = \{\sharp, \natural\},\ q_0 = \sharp,\ \delta(\sharp) = \langle \to, \natural, \natural \rangle,\ \delta(\natural) = \langle \wedge, \sharp, \sharp \rangle$.

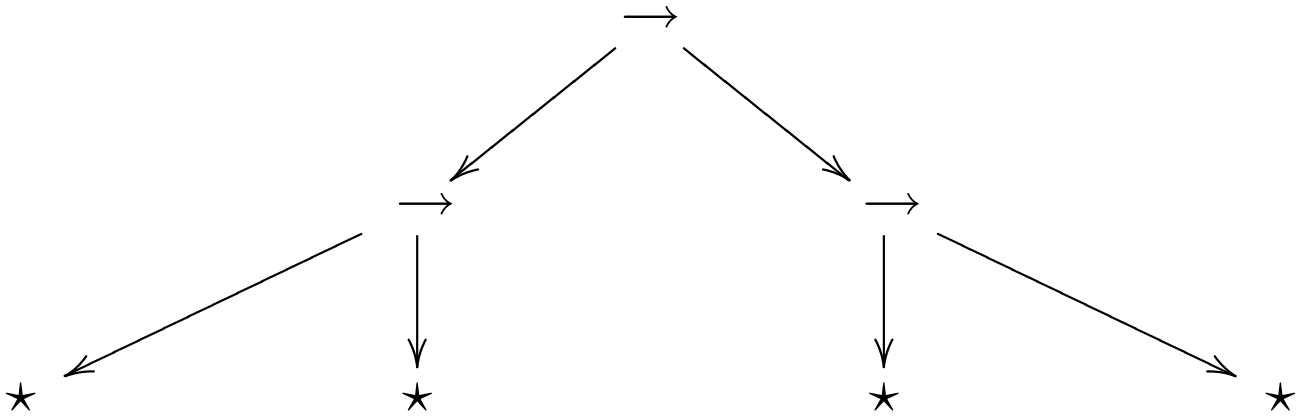
$\star \to \star$の構文木



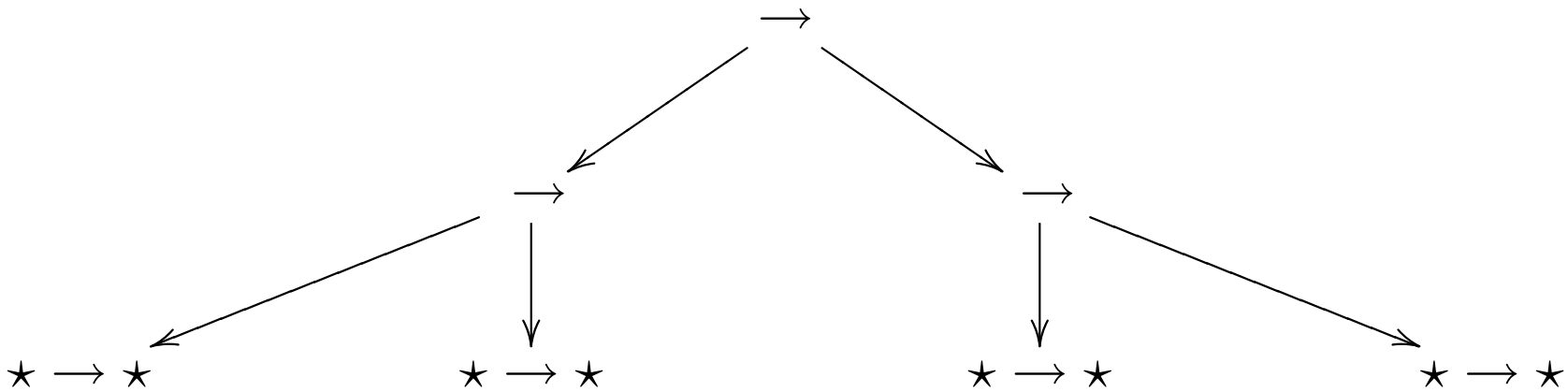
$\star \to \star$の構文木



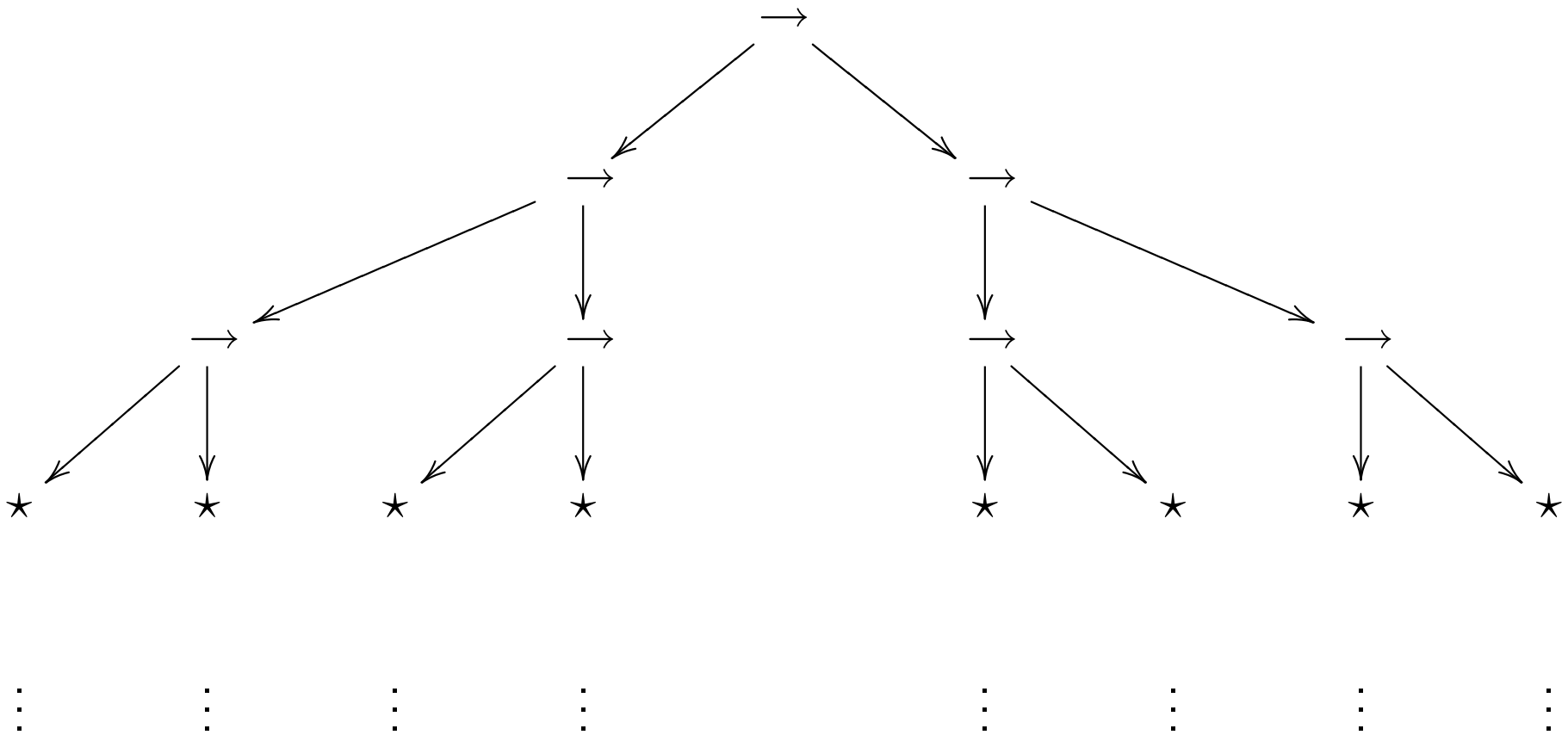
$\star \to \star$の構文木



$\star \to \star$の構文木



$\star \to \star$の構文木



$\star \to \star$を表すマシンの状態遷移図

Definition

$R \subseteq T_\Sigma \times T_\Sigma$ が T_Σ 上の **bisimulation** であるのはすべての $\mathcal{M}, \mathcal{M}' \in T_\Sigma$ に対して

$$\mathcal{M} R \mathcal{M}' \Rightarrow \begin{cases} \mathbf{head}(\mathcal{M}) = \mathbf{head}(\mathcal{M}') \\ \mathbf{tail}_n(\mathcal{M}) R \mathbf{tail}_n(\mathcal{M}') \end{cases}$$

が成り立つときである.

余論理式 $\mathcal{M}, \mathcal{M}' \in T_\Sigma$ が **bisimilar** であると言われるのは，ある bisimulation R に対して $\mathcal{M} R \mathcal{M}'$ が成り立つときである.

bisimilar な二つの余論理式は観察に対して等しい，すなわち私たちはそれらの振る舞いを観察することによって区別することが出来ない.

例

次の二つの余論理式は bisimilar である.

- $\mathcal{M} = (\{q\}, q, \delta)$, ただし $\delta(q) = (\to, q, q)$
- $\mathcal{M}' = (\{q', q''\}, q', \delta')$, ただし $\delta'(q') = (\to, q'', q'')$, $\delta'(q'') = (\to, q', q')$.

論理式と余論理式の対は，同種の双対的な概念の一例である．

$$
\begin{array}{ccc}
\text{代数} & \Longleftrightarrow & \text{余代数} \\
\text{帰納法} & \Longleftrightarrow & \text{余帰納法} \\
\text{リスト} & \Longleftrightarrow & \text{ストリーム} \\
\text{論理式} & \Longleftrightarrow & \text{余論理式}
\end{array}
$$

従って余論理式について考えるのは自然なステップであるように思われる．

しかし以下の質問もまた自然である．

- それは何らかの意味を持つのか？
- それは何らかの文脈で有用なのか？

—「然り」

- 余論理式の一部は**再帰型**と見なすことが出来る.
- それらは**論理と計算の間のギャップ**を埋めることが出来る.

1 余論理式

2 論理と計算の関係

3 再帰型としての余論理式

4 証明体系

5 循環的言語としての余論理式

Curry-Howard 同型

論理		型付き λ 計算
命題論理	$\Longleftrightarrow$	単純型付き λ 計算
一階述語論理	$\Longleftrightarrow$	依存型理論
二階命題論理	$\Longleftrightarrow$	多態型理論
	etc.	
命題	$\Longleftrightarrow$	型
$\rightarrow$-I ($\forall$-I)	$\Longleftrightarrow$	抽象
$\rightarrow$-E ($\forall$-E)	$\Longleftrightarrow$	関数適用
証明	$\Longleftrightarrow$	項
正規化	$\Longleftrightarrow$	縮約
	etc.	

Howard Curry 靴店

Howard Curry Shoes

Home of the Talking Tree

HOME | ABOUT | WOMEN | MEN | KIDS | ACCESSORIES | AWARDS | UPCOMING EVENTS | CONTACT US

SAVE UP TO $70

at our

END·OF·SUMMER·SALE

50% OFF ALL SANDALS 20% OFF ALL ATHLETIC SHOES

On Mens, Womens and Kids >>

論理と計算の違い

かくして次のクリシェが生まれた：

「証明を構成することはプログラムを書くことである」

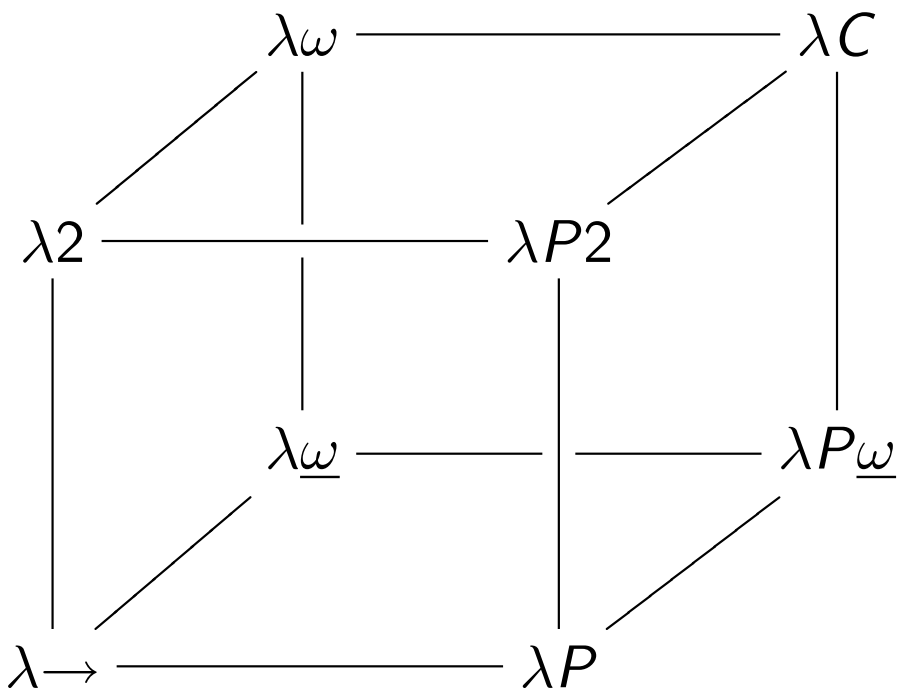
しかしこの逆は真ではない：

「プログラムを書くことは証明を構成すること**ではない**」

Cf. Nordström et al. *Programming in Martin-Löf Type Theory*:

> 型理論においては，正しいことが証明できるプログラムを開発するとともに，プログラミングのタスクの仕様を書くことが出来る．従って型理論はプログラミング言語以上のものであって，プログラミング言語ではなく，むしろ *LCF* や *PL/CV* のような形式化されたプログラミング論理と比較されるべきである．

ラムダ・キューブ



これらはすべて強正規化可能であり，従って Turing 完全ではない.

一方，実際のプログラミング言語のほとんどは強正規性を持たない.

停止しないプログラムの例：

```
while (now.isToday()) {
    System.out.print(  );
}
```


この **while** のような構文は特に再帰的関数の定義のために必要である.

以下の定義がどのように働くか考えよう：

$$f(n) = \begin{cases} 1 & \text{if n} = 0 \\ n * f(n-1) & \text{otherwise} \end{cases}$$

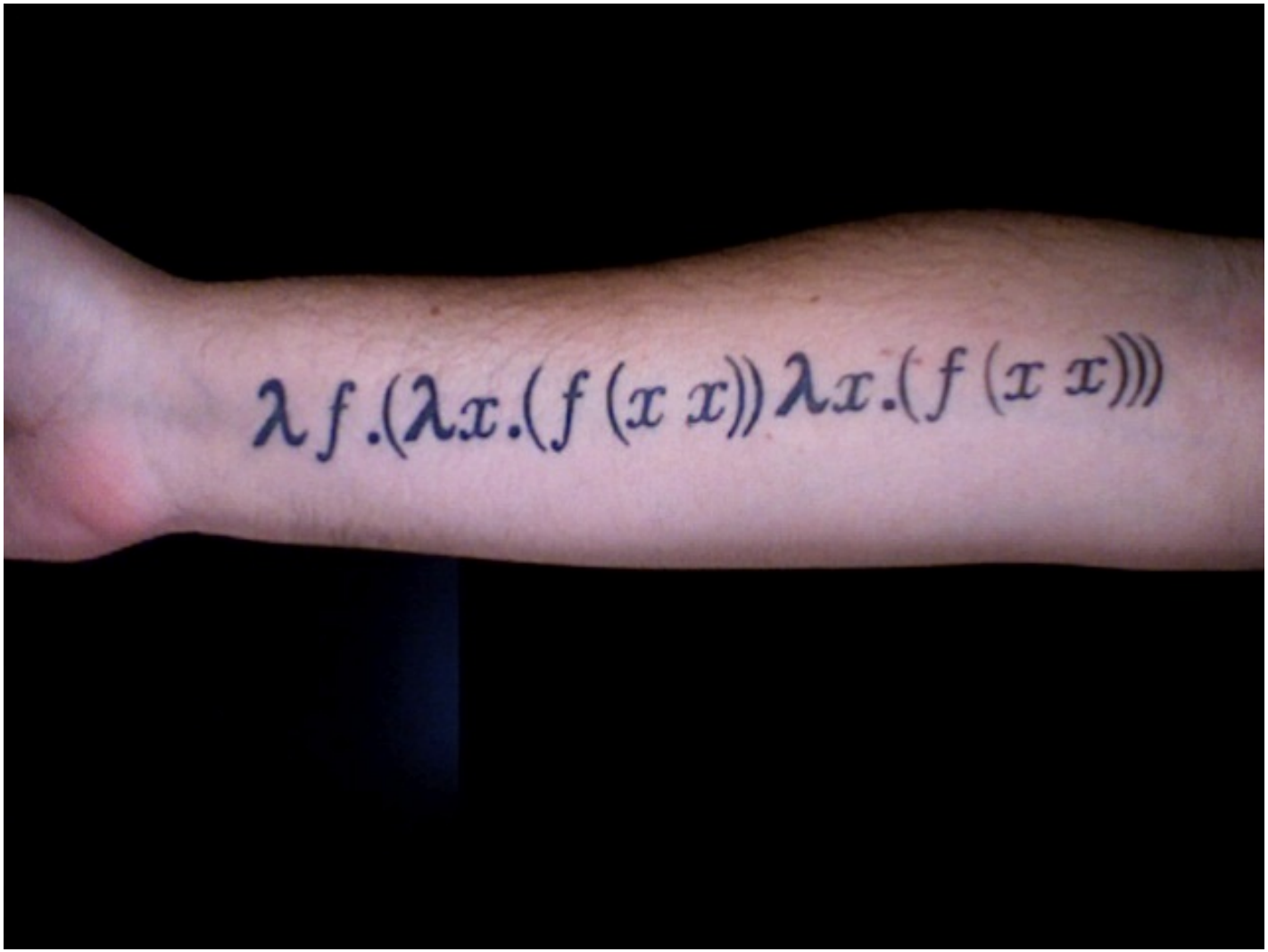
ここで行っているのは以下のように定義されるファンクショナル $\Phi : (N \rightharpoonup N) \to (N \rightharpoonup N)$ の最小不動点を求めることである，と言われる：

$$\Phi(f)(n) = \begin{cases} 1 & \text{if n} = 0 \\ n * f(n-1) & \text{otherwise} \end{cases}$$

ただし $N \rightharpoonup N$ は自然数上の部分関数の集合.

しかしこのようなファンクショナルの最小不動点はどのようにして求められるのか？

Curry の不動点コンビネータ



Definition

$x \in A$ が関数 $f : A \to A$ の**不動点**であるのは $f(x) = x$ が成り立つときである.
$F : (A \to A) \to A$ が**不動点演算子**であるのはすべての $f : A \to A$ に対して $f(F(f)) = F(f)$ が成り立つときである.

Y を $\lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$ としよう. このとき任意の項 M に対して

$$\begin{aligned} YM &\to_\beta M((\lambda x.M(xx))(\lambda x.M(xx))) \\ &\to_\beta M(M((\lambda x.M(xx))(\lambda x.M(xx)))) \\ M(YM) &\to_\beta M(M((\lambda x.M(xx))(\lambda x.M(xx)))) \end{aligned}$$

$\therefore M(YM) =_\beta YM.$

不動点演算子を使った階乗の定義（Scheme で）：

```
(define Y
    (lambda (f)
        ((lambda (x) (f (x x))) (lambda (x) (f (x x))))))

(define (F f)
    (lambda (n)
        (if (= n 0) 1
            (* n (f (- n 1))))))

((Y F) 5)

>> 120
```

以下の事実に注意

Fact

- Y は β 正規形を持たない，よって強正規性を持ついかなる純粋型システム $\lambda\star$ に対しても $\not\vdash_{\lambda\star} Y : \sigma$.
- Y（もしくは任意の項の最小不動点を計算する何らかの他のメカニズム）を備えた $\lambda\!\!\rightarrow$ は *Turing* 完全である.

従ってそのようなメカニズムが論理（型付き λ 計算）とプログラミング言語を分ける大きな要因の一つである.

1 余論理式

2 論理と計算の関係

3 再帰型としての余論理式

4 証明体系

5 循環的言語としての余論理式

反射的領域

Definition

型 D が**反射的領域**であるのは，項 $\phi : D \to (D \to D)$ と
$\psi : (D \to D) \to D$ が存在して

$$\phi \circ \psi = id_{D \to D}$$

が成り立つときである．ただしここで $id_{D \to D} : (D \to D) \to (D \to D)$ は
$(D \to D)$ 上の恒等関数.

反射的領域

Definition

反射的領域に対する規則：

$$\frac{M : D \to D}{\psi M : D}\ \text{fold} \qquad \frac{M : D}{\phi M : D \to D}\ \text{unfold}$$

$$\frac{M : D \to D}{\phi(\psi M) = M : D \to D}\ \text{reflexive identity}$$

Yコンビネータの構成

Y combinator

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x : D}{\phi x : D \to D}\,\text{unfold} \qquad x : D}{\phi x x : D}}{y(\phi x x) : D}}{\lambda x^D.y(\phi x x) : D \to D} \qquad \dfrac{\dfrac{\dfrac{\dfrac{x : D}{\phi x : D \to D}\,\text{unfold} \qquad x : D}{\phi x x : D}}{y(\phi x x) : D}}{\lambda x^D.y(\phi x x) : D \to D}}{(\lambda x^D.y(\phi x x))(\psi(\lambda x^D.y(\phi x x))) : D}\,\text{fold}}{\lambda y^{D \to D}.(\lambda x^D.y(\phi x x))(\psi(\lambda x^D.y(\phi x x))) : (D \to D) \to D}$$

反射的領域はより一般的な再帰型の概念によって定義することが出来る.
再帰型とは

$$\textbf{rec}\ t.\tau$$

という形式の型表現である．ただし τ は型表現，t は型変数である.

大雑把に言って $\textbf{rec}\ t.\tau$ は次の等式を満たす型表現である.

$$\textbf{rec}\ t.\tau = \tau[\textbf{rec}\ t.\tau/t]$$

反射的領域 D は $\textbf{rec}\ t.t \to t$ によって定義される.

型表現

Definition

型変数の集合 **TVar** が与えられているものとする（t によって任意の型変数を表す）．型表現の集合 **TExp** は以下の文法で定義される：

$$\tau ::= t \mid (\sigma, \tau_1, \ldots, \tau_n) \mid \mathbf{rec}\ t.\tau$$

ただし $\sigma \in \Sigma_n$.

Definition

型表現が**余論理式的**であるのは，それが閉じていてかつ $\mathbf{rec}\ t.t'$ という部分表現を含まないときである（$t = t'$ であるか否かを問わず）．余論理式的な表現の集合を $\mathbf{TExp}_0$ によって表す.

有限の余論理式を余論理式的型表現に変換させる関数 $F : T_{\Sigma}^{\mathsf{fin}} \to \mathbf{TExp}_0$ を以下のように定義する：

有限の余論理式 $\mathcal{M} = \langle Q, q_0, \delta \rangle$ を考える．$Q = \{q_k : 0 \leq k \leq n\}$ としよう．

$$S_k = (\sigma_k, t_{i(k,1)}, \ldots, t_{i(k,m_k)})$$

とする．ただしここで $\delta(q_k) = (\sigma_k, q_{i(k,1)}, \ldots, q_{i(k,m_k)})$ であり，各 $0 \leq k \leq n, 0 \leq j \leq m_k$ に対して $t_{i(k,j)}$ は異なる型変数とする．

$\hat{S}_k (0 \leq k \leq n)$ を帰納的に以下のように定義する：

$$\begin{aligned} \hat{S}_n &= \textbf{rec}\ t_n.S_n \\ \hat{S}_{k-1} &= \textbf{rec}\ t_{k-1}.(S_{k-1}[\hat{S}_n/t_n] \ldots [\hat{S}_k/t_k]) \end{aligned}$$

$\hat{S}_0$ を $F\mathcal{M}$ の値とする.

例えば次の余論理式 $\mathcal{M} = \langle Q, q_0, \delta \rangle$ を考えよう：

$$\begin{aligned} Q &= \{q_0, q_1, q_3\}, \\ \delta(q_0) &= \langle \to, q_1, q_2 \rangle, \\ \delta(q_1) &= \langle \wedge, q_2, q_0 \rangle, \\ \delta(q_2) &= \langle \vee, q_0, q_1 \rangle. \end{aligned}$$

F を $\mathcal{M}$ に適用することで，次が得られる：

$$\textbf{rec}\ t_0.(\textbf{rec}\ t_1.(\textbf{rec}\ t_2.(t_0 \vee t_1) \wedge t_0) \to \textbf{rec}\ t_2.(t_0 \vee \textbf{rec}\ t_1.(\textbf{rec}\ t_2.(t_0 \vee t_1) \wedge t_0)))$$

$$
\begin{aligned}
S_2 &= t_0 \vee t_1 \\
S_1 &= t_2 \wedge t_0 \\
S_0 &= t_1 \to t_2
\end{aligned}
$$

$$
\begin{aligned}
\hat{S}_2 &= \mathbf{rec}\ t_2.(t_0 \vee t_1) \\
S_1[\hat{S}_2/t_2] &= \mathbf{rec}\ t_2.(t_0 \vee t_1) \wedge t_0 \\
S_0[\hat{S}_2/t_2] &= t_1 \to \mathbf{rec}\ t_2.(t_0 \vee t_1)
\end{aligned}
$$

$$
\begin{aligned}
\hat{S}_1 &= \mathbf{rec}\ t_1.(\mathbf{rec}\ t_2.(t_0 \vee t_1) \wedge t_0) \\
S_0[\hat{S}_2/t_2][\hat{S}_1/t_1] &= \mathbf{rec}\ t_1.(\mathbf{rec}\ t_2.(t_0 \vee t_1) \wedge t_0) \to \\
&\qquad \mathbf{rec}\ t_2.(t_0 \vee \mathbf{rec}\ t_1.(\mathbf{rec}\ t_2.(t_0 \vee t_1) \wedge t_0))
\end{aligned}
$$

$$
\begin{aligned}
\hat{S}_0 &= \mathbf{rec}\ t_0.(\mathbf{rec}\ t_1.(\mathbf{rec}\ t_2.(t_0 \vee t_1) \wedge t_0) \to \\
&\qquad \mathbf{rec}\ t_2.(t_0 \vee \mathbf{rec}\ t_1.(\mathbf{rec}\ t_2.(t_0 \vee t_1) \wedge t_0)))
\end{aligned}
$$

注意

S_k の順序が結果に影響するかもしれない．例えば上記の $\hat{S}_0$ の構成において，S_2 ではなく S_1 から変換の手続きを始めるならば結果は

$$\mathbf{rec}\ t_0.(\mathbf{rec}\ t_1.(\mathbf{rec}\ t_2.(t_0 \vee \mathbf{rec}\ t_1.(t_2 \wedge t_0)) \wedge t_0) \to \mathbf{rec}\ t_2.(t_0 \vee \mathbf{rec}\ t_1.(t_2 \wedge t_0)))$$

であり

$$\mathbf{rec}\ t_0.(\mathbf{rec}\ t_1.(\mathbf{rec}\ t_2.(t_0 \vee t_1) \wedge t_0) \to \mathbf{rec}\ t_2.(t_0 \vee \mathbf{rec}\ t_1.(\mathbf{rec}\ t_2.(t_0 \vee t_1) \wedge t_0)))$$

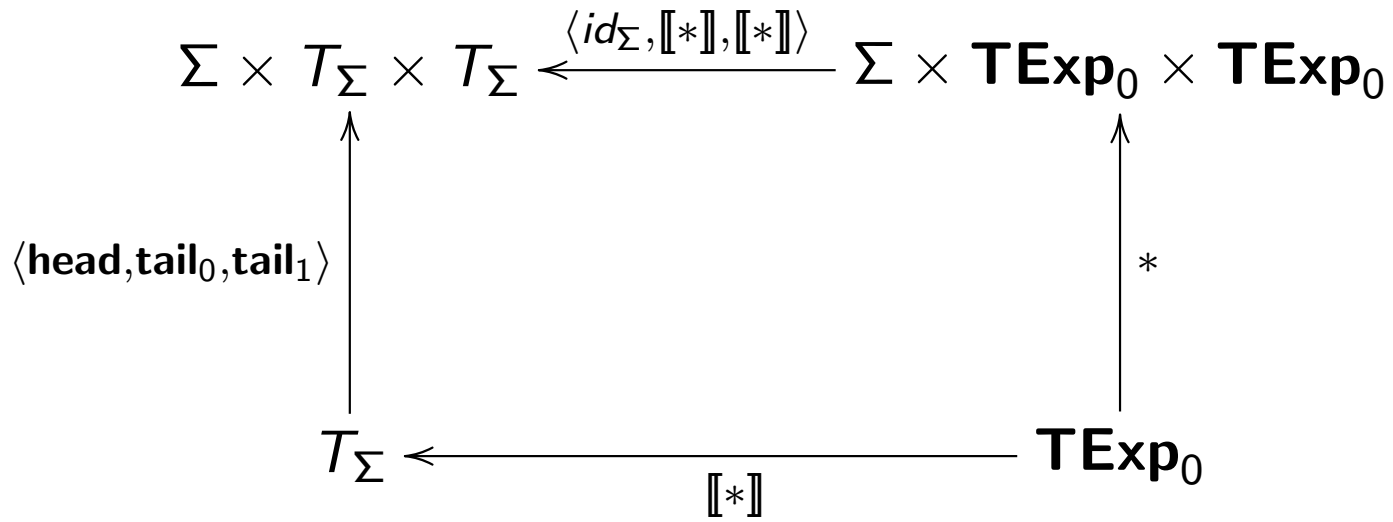
ではない．しかしこの違いは後に述べる理由から無視できる．

余論理式的型表現を有限の余論理式に変換する手続き $G : \mathbf{TExp}_0 \to T_\Sigma^{\mathsf{fin}}$ を定義しよう.

$* : \mathbf{TExp}_0 \to \coprod_{n\geq 0} \Sigma \times (\mathbf{TExp}_0)^n$ を次のように定義する：

$$\begin{aligned}(\sigma, \tau_1, \ldots, \tau_n)^* &= (\sigma, \tau_1, \ldots, \tau_n) \\ (\mathbf{rec}\ t.\tau)^* &= (\tau[\mathbf{rec}\ t.\tau/t])^*\end{aligned}$$

T_Σ は終対象なので，以下のダイアグラムを可換にする
$[\![*]\!] : \mathbf{TExp}_0 \to T_\Sigma$ が一意的に存在する：



この $[\![*]\!]$ を G と呼ぼう.

Lemma

S_0, S_1, S_2 は t_0, t_1, t_3 のみを型変数として含む型表現であり，**rec** を含まないものとする．$\overline{\theta}$ は（空かもしれない）連続的置換とし，その構成要素の各々は $[\mathbf{rec}\ t_i.S_i\overline{\theta'}/t_i]$ という形式であるとする．このとき
$G(\mathbf{rec}\ t_0.S_0[\mathbf{rec}\ t_1.S_1\overline{\theta}_1/t_1][\mathbf{rec}\ t_2.S_2\overline{\theta}_2/t_2])$ と
$G(\mathbf{rec}\ t_0.S_0[\mathbf{rec}\ t_1.S_1\overline{\theta'_1}/t_1][\mathbf{rec}\ t_2.S_2\overline{\theta'_2}/t_2])$ は *bisimilar* である．

証明．通常の余帰納法による． □

この結果によって上記の注意が正当化される．

Proposition

任意の $\mathcal{M} \in T_{\Sigma}^{fin}$ に対して，$\mathcal{M}$ と $GF(\mathcal{M})$ は *bisimilar* である．

証明．上記の補題を用いて，余帰納法で示される． □

上記の結果に基づいて，私たちは余論理式を $T_{\Sigma}^{\mathsf{fin}}$ に制限することを提案する．

この制限には以下のような利点がある：

- 有限の手段で捉えられないような余論理式を排除する．
- 計算とプログラミング言語との関わりで重要性を持つ余論理式のみを考えることが出来る．
- 既に存在している意味論と証明システムを利用できる．

1 余論理式

2 論理と計算の関係

3 再帰型としての余論理式

4 証明体系

5 循環的言語としての余論理式

Abramsky “Domain theory in logical form” (1991) は領域理論の言語の「論理的解釈」を定式化している.

この言語には，$\to$（関数空間），$\times$（直積），$\oplus$（融合積），$\mathscr{P}$（Plotkin 冪領域），**rec**（再帰型）などの，領域理論における通常の型コンストラクタが含まれる.

私たちはこの言語を余論理式の証明のために利用することが出来る.

証明体系

単純性のために $\Sigma = \{\to, \wedge\}$ としよう.

命題論理の自然演繹に加えて，各余論理式 $\mathcal{M} = \langle Q, q_0, \delta \rangle$ に対して次の規則を採用する：

$$\frac{M : \delta(q)}{\mathbf{fold}_q^{\mathcal{M}}(M) : q}\ \textbf{rec-I} \qquad \frac{M : q}{\mathbf{unfold}_q^{\mathcal{M}}(M) : \delta(q)}\ \textbf{rec-E}$$

私たちは $\delta(q)$ が (σ, q', q'') を表すものとして記号を乱用している．ただしここで $\delta(q) = (\sigma, q', q'')$.

$\mathcal{M}$ が証明されるのはその初期状態が証明されるときである.

例

$Q = \{q_0, q_1, q_2\}$
$\delta(q_0) = q_1 \to q_2,\ \delta(q_1) = q_2 \to q_0,\ \delta(q_2) = q_0 \to q_1,$

$$\dfrac{\dfrac{\dfrac{\dfrac{(x : q_1)}{\lambda y.x : q_0 \to q_1}\,\to\text{-I}}{\mathbf{fold}(\lambda y.x) : q_2}\,\mathbf{rec\text{-}I}}{\lambda x.\mathbf{fold}(\lambda y.x) : q_1 \to q_2}\,\to\text{-I}}{\mathbf{fold}(\lambda x.\mathbf{fold}(\lambda y.x)) : q_0}\,\mathbf{rec\text{-}I}$$

例

Curry のパラドクス

$Q = \{q_0, q_1\}$
$\delta(q_0) = q_0 \wedge q_0,\ \delta(q_1) = q_1 \to q_0$

$$\dfrac{\dfrac{\dfrac{(x : q_1)}{\mathbf{unfold}_{q_1}(x) : q_1 \to q_0}\,\mathbf{rec\text{-}E} \qquad (x : q_1)}{\mathbf{unfold}_{q_1}(x)x : q_0}}{\lambda x^{q_1}.(\mathbf{unfold}_{q_1}(x)x) : q_1 \to q_0}\,x \qquad \dfrac{\dfrac{\dfrac{(x : q_1)}{\mathbf{unfold}_{q_1}(x) : q_1 \to q_0}\,\mathbf{rec\text{-}E} \qquad (x : q_1)}{\mathbf{unfold}_{q_1}(x)x : q_0}}{\lambda x^{q_1}.(\mathbf{unfold}_{q_1}(x)x) : q_1 \to q_0}\,x}{\mathbf{unfold}_{q_1}(\lambda x^{q_1}.(\mathbf{unfold}_{q_1}(x)x)) : q_1}\,\mathbf{rec\text{-}I}$$
$$(\lambda x^{q_1}.(\mathbf{unfold}_{q_1}(x)x))(\mathbf{unfold}_{q_1}(\lambda x^{q_1}.(\mathbf{unfold}_{q_1}(x)x))) : q_0$$

注意

- この体系は有限の余論理式が自己言及的な文として理解され得るということを示唆する.
- 証明に際しては，証明の外で余論理式を特定する必要がある．そのためこの体系は純粋に構文論的でないように思われるかもしれない. しかし有限の余論理式に関する限り，この特定は構文論的な仕方で出来る（それは **let** を使った構文と同じようなものである）.
- 証明の中には正規化できないものがある.
- 余論理式の間の bisimulation 関係は必ずしも証明可能を保存しない.

1 余論理式

2 論理と計算の関係

3 再帰型としての余論理式

4 証明体系

5 循環的言語としての余論理式

循環的言語

- Leitgeb and Hieke “Circular languages” (2004) は「自己適用的表現」と「自己包含的表現」を許すような言語に対する公理系を与えている.
- 彼らはここで循環的でない言語の公理系 AppI と循環的な言語の公理系 AppCI を定式化する.
- 余論理式は AppCI のモデルになる.

公理系 Appl

Appl は語彙として

- アトムの集合 $\{a_i\}_{i \in Ind}$,
- 2 項関数記号 **seq**，**app**,
- 1 項述語 **atomic**，**object**，**formula**

を持つ．直観的に言って **seq** は対象から列を形成し，**app** は式を対象に適用する操作を表す．これらの結果はともに **object** になるが，**formula** になるのは **app** の結果だけである．

公理系 AppI

Identity Axioms:

SEQ-IDENTITY:
$\forall x, y, u, v(\mathbf{seq}(x, y) = \mathbf{seq}(u, v) \rightarrow x = u \wedge y = v).$

APP-IDENTITY:
$\forall x, y, u, v(\mathbf{app}(x, y) = \mathbf{app}(u, v) \rightarrow x = u \wedge y = v).$

公理系 AppI

Categorization Axioms:

ATOMIC-DISJOINT-SEQ:
$\forall x(\mathbf{atomic}(x) \rightarrow \neg\exists y, z(x = \mathbf{seq}(y, z)))$.

ATOMIC-DISJOINT-APP:
$\forall x(\mathbf{atomic}(x) \rightarrow \neg\exists y, z(x = \mathbf{app}(y, z)))$.

SEQ-DISJOINT-APP:
$\forall x, y, u, v(\mathbf{seq}(x, y) \neq \mathbf{app}(u, v))$.

公理系 AppI

Atomic Axiom (Scheme):

ATOMS:
$\mathbf{atomic}(a_i) \wedge a_i \neq a_j.$

公理系 AppI

Structural Axioms:

OBJECTS:

$$\forall z$$
$$((\mathbf{atomic}(z) \rightarrow \mathbf{object}(z)) \wedge$$
$$\forall x, y(\mathbf{formula}(x) \wedge \mathbf{object}(y) \wedge z = \mathbf{seq}(x, y) \rightarrow \mathbf{object}(z)) \wedge$$
$$\forall x, y(\mathbf{formula}(x) \wedge \mathbf{object}(y) \wedge z = \mathbf{app}(x, y) \rightarrow \mathbf{object}(z))).$$

FORMULAS:

$$\forall z$$
$$((\mathbf{atomic}(z) \rightarrow \mathbf{formula}(z)) \wedge$$
$$\forall x, y(\mathbf{formula}(x) \wedge \mathbf{object}(y) \wedge z = \mathbf{app}(x, y) \rightarrow \mathbf{formula}(z))).$$

公理系 AppI

Induction Axiom (Scheme):

INDUCTION:

$$
\begin{aligned}
&\forall z \\
&\quad ((\mathbf{atomic}(z) \to \Phi[z]) \wedge \\
&\quad (\mathbf{atomic}(z) \to \Psi[z] \wedge \\
&\quad \forall x, y(\Psi[x] \wedge \Phi[y] \wedge z = \mathbf{seq}(x, y) \to \Phi[z]) \wedge \\
&\quad \forall x, y(\Psi[x] \wedge \Phi[y] \wedge z = \mathbf{app}(x, y) \to \Phi[z]) \wedge \\
&\quad \forall x, y(\Psi[x] \wedge \Phi[y] \wedge z = \mathbf{app}(x, y) \to \Psi[z])) \\
&\to \forall z((\mathbf{object}(z) \to \Phi[z]) \wedge (\mathbf{formula}(z) \to \Psi[z])).
\end{aligned}
$$

（Φ と Ψ は任意の一階の式を表す）

公理系 AppI

- 公理系 AppI は自己適用の任意の事例——例えば $P(P)$——を含んでいる.
- しかし AppI は自己包含の事例——例えば $\alpha = P(\alpha)$——を含まない.
- そこで帰納的な対象と式の構成についての公理を余帰納的な構成に置き換えた公理系 AppCI が考えられる.

公理系 AppCI

Identity Axioms，Categorization Axioms，Atom Axiom は AppI と同じ．
Structural Axioms は次のものに置き換える．

Structural Axioms**:*

OBJECTS*:

$$\forall z(\mathbf{object}(z) \rightarrow (\mathbf{atomic}(z) \vee \exists x, y(\mathbf{formula}(x) \wedge \mathbf{object}(y) \wedge z = \mathbf{seq}(x, y)) \vee \exists x, y(\mathbf{formula}(x) \wedge \mathbf{object}(y) \wedge z = \mathbf{app}(x, y)))).$$

FORMULAS*:

$$\forall z(\mathbf{formula}(z) \rightarrow (\mathbf{atomic}(z) \vee \exists x, y(\mathbf{formula}(x) \wedge \mathbf{object}(y) \wedge z = \mathbf{app}(x, y)))).$$

公理系 AppCI

Induction Axiom は次のものに置き換える.

Coinduction Axiom (Scheme):

CO-INDUCTION:

$$
\begin{aligned}
&\forall z \\
&\quad ((\Phi[z] \rightarrow \\
&\qquad (\mathbf{atomic}(z) \vee \\
&\qquad \exists x, y(\Psi[x] \wedge \Phi[y] \wedge z = \mathbf{seq}(x, y)) \vee \\
&\qquad \exists x, y(\Psi[x] \wedge \Phi[y] \wedge z = \mathbf{app}(x, y)))) \\
&\quad \wedge \\
&\quad (\Psi[z] \rightarrow \\
&\qquad (\mathbf{atomic}(z) \vee \\
&\qquad \exists x, y(\Psi[x] \wedge \Phi[y] \wedge z = \mathbf{app}(x, y))))) \\
&\rightarrow \forall z((\Phi[z] \rightarrow \mathbf{object}(z)) \wedge (\Psi[z] \rightarrow \mathbf{formula}(z))).
\end{aligned}
$$

公理系 AppICI

- AppCI はそのままでは T_Σ をモデルとして持たない.
- AppCI を修正した公理系 AppICI を考える.

公理系 AppICI

AppICI は語彙として

- アトムの集合 $\{\bot\} \cup \{a_{nk} : n, k \in N\}$,
- 2 項関数記号 **seq**，**app**,
- 1 項述語 $\mathbf{atomic}_{\bot}$，$\mathbf{atomic}_n$, **object**，$\mathbf{object}_n$（ただし $n \in N$）, **formula**

を持つ.

公理系 AppICI

Structural Axioms†*:*

OBJECTS†:

$\forall z((\mathbf{atomic}_{\perp}(z) \rightarrow \mathbf{object}(z)) \wedge$
$\forall x, y(\mathbf{formula}(x) \wedge \mathbf{object}(z) \wedge z = \mathbf{seq}(x, y) \rightarrow \mathbf{object}(z))).$

OBJECTS$_n$:

$\forall z(\mathbf{object}_0(z) \leftrightarrow \mathbf{atomic}_{\perp}(z)) \wedge$
$\forall z(\mathbf{object}_{n+1}(z) \leftrightarrow$
$\mathbf{object}(z) \wedge \exists x, y(\mathbf{formula}(x) \wedge \mathbf{object}_n(y) \wedge z = \mathbf{seq}(x, y))).$

FORMULAS†:

$\forall z(\mathbf{formula}(z) \rightarrow$
$\exists x, y \exists n \in N(\mathbf{atomic}_n(x) \wedge \mathbf{object}_n(y) \wedge z = \mathbf{app}(x, y))).$

公理系 AppICI

Induction Axiom (Scheme)†*:*

$$
\begin{aligned}
&\textsc{Induction}^{\dagger}:\\
&\forall z\\
&\quad ((\mathbf{atomic}_{\perp}(z) \rightarrow \Phi[z]) \wedge\\
&\quad \forall x, y(\mathbf{formula}(x) \wedge \Phi(z) \wedge z = \mathbf{seq}(x, y) \rightarrow \Phi[z]))\\
&\rightarrow \forall z(\mathbf{object}(z) \rightarrow \Phi(z)).
\end{aligned}
$$

公理系 AppICI

CoInduction Axiom は次のものに置き換える.
Coinduction Axiom (Scheme)†*:*

$$
\begin{aligned}
&\textsc{Co-Induction}^{\dagger}:\\
&\forall z\\
&\quad (\Psi[z] \rightarrow\\
&\qquad (\mathbf{atomic}_0(z) \vee\\
&\qquad \exists x, y \exists n \in N(\mathbf{atomic}_n(x) \wedge \Phi(y) \wedge z = \mathbf{app}(x, y))))\\
&\rightarrow \forall z(\Psi[z] \rightarrow \mathbf{formula}(z)).
\end{aligned}
$$

AppICI のモデル

- T_Σ が AppICI のモデルになることは自明.
- さらに T_Σ は次の公理を満たす.

  $\textsc{Solution}^\dagger$:
  $\exists x_1, x_2, \ldots, x_n$
  $(\mathbf{formula}(x_1) \wedge \cdots \wedge \mathbf{formula}(x_n) \wedge$
  $x_1 = f_1[x_1, x_2, \ldots, x_n] \wedge \cdots \wedge x_n = f_n[x_1, x_2, \ldots, x_n])$

  ただし $f_1 \ldots f_n \in \mathit{FormulaTerms}^\dagger$. $\mathit{FormulaTerms}^\dagger$ は次のように定義される.

$\mathit{ObjectTerms}^{\dagger}$ / $\mathit{FormulaTerms}^{\dagger}$

1. 変数は $\mathit{ObjectTerms}_1$ と $\mathit{FormulaTerms}$ の両方に属する.
2. $a_{\bot} \in \mathit{ObjectTerms}_0$, $a_{0k} \in \mathit{FormulaTerms}$.
3. $f \in \mathit{FormulaTerms}, o \in \mathit{ObjectTerms}_n \Rightarrow \langle f, o \rangle \in \mathit{ObjectTerms}_{n+1}$.
4. $o \in \mathit{ObjectTerms}_n \Rightarrow \langle a_{nk} : o \rangle \in \mathit{FormulaTerms}$.

まとめ

- 論理学とプログラミングの間には Curry-Howard 同型が成り立つが，しかし大きなギャップもある．
- それは不動点演算子のようなメカニズムを持てるかどうかという点である．
- 余論理式はプログラミング言語の再帰型に対応するものであり，このギャップを埋めるものである．
- そのようなものとして，余論理式を有限なものに制限することは理にかなっている．
- そのときある種の証明体系が余論理式にも与えることが出来る．
- 余論理式はまた循環的言語のモデルの一種と捉えることも出来る．